Information-Theoretic Broadcast with Dishonest Majority for Long Messages

Wutichai Chongchitmate[1(✉)] and Rafail Ostrovsky[2]

[1] Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University, Bangkok, Thailand
wutichai.ch@chula.ac.th

[2] Department of Computer Science and Department of Mathematics, University of California, Los Angeles, Los Angeles, CA, USA
rafail@cs.ucla.edu

Abstract. Byzantine broadcast is a fundamental primitive for secure computation. In a setting with n parties in the presence of an adversary controlling at most t parties, while a lot of progress in optimizing communication complexity has been made for $t < n/2$, little progress has been made for the general case $t < n$, especially for information-theoretic security. In particular, all information-theoretic secure broadcast protocols for ℓ-bit messages and $t < n$ and optimal round complexity $\mathcal{O}(n)$ have, so far, required a communication complexity of $\mathcal{O}(\ell n^2)$. A *broadcast extension* protocol allows a long message to be broadcast more efficiently using a small number of single-bit broadcasts. Through broadcast extension, so far, the best achievable round complexity for $t < n$ setting with the optimal communication complexity of $\mathcal{O}(\ell n)$ is $\mathcal{O}(n^4)$ rounds.

In this work, we construct a new broadcast extension protocol for $t < n$ with information-theoretic security. Our protocol improves the round complexity to $\mathcal{O}(n^3)$ while maintaining the optimal communication complexity for long messages. Our result shortens the gap between the information-theoretic setting and the computational setting, and between the optimal communication protocol and the optimal round protocol in the information-theoretic setting for $t < n$.

W. Chongchitmate—Work done while the author was at Department of Computer Science, University of California, Los Angeles.

R. Ostrovsky—Research supported in part by NSF grant 1619348, DARPA SafeWare subcontract to Galois Inc., DARPA SPAWAR contract N66001-15-1C-4065, US-Israel BSF grant 2012366, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. The views expressed are those of the authors and do not reflect position of the Department of Defense or the U.S. Government.

A. Beimel and S. Dziembowski (Eds.): TCC 2018, LNCS 11239, pp. 370–388, 2018.
https://doi.org/10.1007/978-3-030-03807-6_14

1 Introduction

A (Byzantine) broadcast protocol allows a party, called "sender," to distribute a message among n parties such that (1) all honest parties receive the same message, and (2) if the sender is honest, the received message is indeed sent from the sender. This guarantee holds even in the presence of a malicious adversary corrupting up to t parties, possibly including the sender. The adversary controls the behavior of the corrupted parties and may divert from the protocol.

Broadcast is one of the most fundamental primitives used in cryptographic protocols—especially secure multi-party computation (MPC). Most MPC protocols assume broadcast is given by default. However, without a specific hardware setup, broadcast must be built from point-to-point communications. While efficient broadcast can be done with an honest majority, the opposite case is much more common in applications.

Although a lot of progress has been made to improve broadcast protocol in the honest majority case, the best-known result for any number of corruptions has not seen any improvement since [4] for computational security and [15] for information-theoretic security.

Traditionally, broadcast protocols are designed for single bits [13]. However, most applications that use broadcast as a subprotocol often broadcast long messages. While any broadcast protocol can be used multiple times in parallel to broadcast messages of any length, it leads to inefficiency, especially in communication complexity.

Broadcast extension protocol, introduced in [16], uses bit broadcast (or broadcast for fixed-length messages) as a subprotocol, similar to oblivious transfer (OT) extension [1,8,10,11]. The goal is to reduce the communication complexity of broadcasting long messages, compared to trivially executing multiple broadcast protocols.

Broadcast with Dishonest Majority. Unlike when the number of corrupted parties $t < n/3$, it has been shown that broadcast for $t < n$ cannot be achieved in the plain model [13]. To circumvent the impossibility result, Dolev and Strong considered the broadcast protocol in the setup model [4]. They implemented broadcast from any public-key signature assuming public-key infrastructure (PKI) for distributing signing and verification keys for the signature scheme. Their protocol achieves the lower bound $\Omega(n)$ on the round complexity, and $\Omega(n^2)$ on the number of messages exchanged.

For the information-theoretic case, Pfitzmann and Waidner introduce the notion of pseudosignature [14], formalizing unconditionally secure signature in [2], to replace the public-key signature in [4]. The resulting protocol [14,15] is in the correlated randomness model where each party holds a random string generated from some joint distribution instead of PKI. Similar to the computational case, this protocol achieves the lower bound on the round complexity and the number of messages exchanged.

In terms of communication complexity, the broadcast protocol of [15] uses $\mathcal{O}(\ell n^2 + n^6\lambda)$ bits of communication, while that of [4] uses $\mathcal{O}(\ell n^2 + n^3\lambda)$ bits to broadcast a message of length ℓ. In both protocols, a sender sends a message and a corresponding signature to every party, who then sign and pass the message to all other parties in the first two rounds.

In fact, [3] shows that any broadcast protocol must communicate at least $\Omega(n^2)$ bits. Thus, to broadcast a message of length ℓ directly using such protocol requires at least $\Omega(\ell n^2)$ bits of communication. To circumvent this limitation, an extension protocol is designed to reduce the multiplicative factor to the length ℓ of the message to lower than n^2 while increasing the part that is independent of ℓ, thus reducing the overall communication complexity when $\ell \gg \lambda$. Since every party must receive the message, the lower bound on the communication complexity is $\Omega(\ell n)$.

Broadcast Extension. While Turpin and Coan [16] introduced the construction of a broadcast protocol for long messages from bit broadcast, their protocol tolerating $t < n/3$ has the communication complexity of $\mathcal{O}(\ell n^2 + n(B(1)))$, where $B(s)$ is the communication complexity of s-bit broadcast. Fitzi and Hirt [5] first showed how to achieve broadcast with communication complexity $\mathcal{O}(\ell n + poly(n, \lambda))$ in an information-theoretic setting tolerating $t < n/2$ with $poly(n, \lambda) = n^3\lambda + nB(n + \lambda)$. Liang and Vaidya [9] later constructed perfectly secure broadcast tolerating $t < n/3$ with communication complexity $\mathcal{O}(\ell n + \sqrt{\ell} n^2 B(1) + n^4 B(1))$, and Patra [12] improved it to $\mathcal{O}(\ell n + n^2 B(1))$. As mentioned earlier, the best result for communication complexity in an information-theoretic setting tolerating $t < n$ is by Hirt and Raykov [7] with communication complexity $\mathcal{O}(\ell n + (n^4 + n^3\lambda)B(1))$ and round complexity $\mathcal{O}(n^4)$. They also constructed another protocol based on collision-resistant hash functions (CRHF) in the same setting with communication complexity $\mathcal{O}(\ell n + (n^2 + n\lambda)B(1))$ and round complexity $\mathcal{O}(n^3)$. The CRHF-based construction is later improved in round complexity by Ganesh and Patra [6] to $\mathcal{O}(n^2)$, while communication complexity slightly increases to $\mathcal{O}(\ell n + (n\lambda + n^3 \log n)B(1))$.

Round Complexity of Broadcast Protocols. While broadcast can be accomplished in constant round with honest majority, [4] shows that a broadcast protocol secure against an adversary corrupting any number of parties requires at least $\mathcal{O}(n)$ rounds. In the $t < n/3$ and $t < n/2$ settings, the broadcast extension protocols achieve optimal constant round complexity similar to that of bit broadcast [6]. [7] first achieved broadcast protocols for ℓ-bit messages using $\mathcal{O}(\ell n)$ communication complexity for $t < n$ with round complexity $\mathcal{O}(n^3)$ for computational security and $\mathcal{O}(n^4)$ for information-theoretic security, respectively. They left an open question:

Are there broadcast protocols with $\mathcal{O}(\ell n)$ communication complexity for $t < n$ with round complexity lower than $\mathcal{O}(n^3)$ for computational security and lower than $\mathcal{O}(n^4)$ for information-theoretic security?

[6] answered the first part of the question: they constructed a computationally secure protocol with communication complexity of $\mathcal{O}(n^2)$. This result still leaves the second part of the open question unsolved.

1.1 Our Results

We construct a broadcast extension protocol in the information-theoretic setting against adversaries corrupting up to $t < n$ parties. Our result improves the current best-known result in the same setting of [7] in round complexity by a multiplicative factor of n while maintaining the same communication complexity. More formally, we obtain the following theorem.

Theorem 1. *Assuming an oracle for broadcasting short messages, there exists a broadcast protocol achieving information-theoretic security in $t < n$ setting for an ℓ-bit message in $\mathcal{O}(n^2)$ rounds by communicating $\mathcal{O}(\ell n + n^3(B(\lambda) + nB(\log n)))$ bits, where $B(l)$ is the communication complexity of broadcasting l bits.*

Thus, combining the above result with the broadcast protocol of [15] gives the following corollary.

Corollary 1. *There exists a broadcast protocol achieving information-theoretic security in $t < n$ setting for an ℓ-bit message in $\mathcal{O}(n^3)$ rounds by communicating $\mathcal{O}(\ell n + n^{10}\lambda)$ bits.*

This result shortens the gap in round complexity between the information-theoretic case and the computational case where $\mathcal{O}(n^2)$ rounds is achieved in [6]. Closing this gap entirely is left as an open question.

1.2 Our Techniques

Block Broadcast. The traditional broadcast protocol of [4] for $t < n$ prevents a corrupted sender from sending different values to different receivers using signature (or pseudosignature for information-theoretic security [14,15]). The receivers then send their signed values to each other. This means in order to broadcast a message m, both m and the corresponding signature need to be sent and received $\mathcal{O}(n^2)$ times. Thus, the communication complexity of broadcasting a message of length ℓ is at least $\mathcal{O}(\ell n^2)$. Similar to the existing broadcast extension protocols in literature [6,7], a sender in our broadcast protocol cuts a long message into multiple blocks. Each block is sent via point-to-point channels—first from the sender, and later from any parties publicly known to hold the block. Then a broadcast protocol for short messages (multiple times, but independent of ℓ) is used to verify the correctness of the blocks using a universal hash function as in [7]. This keeps the multiplicative factor in the communication complexity linear in n instead of n^2. Similar to [7], our protocol processes one block at a time sequentially.

Multi-party Block Sending. In [7], each block is sent between one pair of parties at a time. In order to improve the round complexity, we use the technique in [6] for the computational security setting where a block is sent between multiple pairs of parties at the same time. In each round, a block is sent to every party not holding the block and satisfying a certain condition from a designated party that holds the block and is still trusted by the receiving party. In particular, if all parties are honest, they will all receive a block in one round.

Checking Block Validity. In order to ensure that all honest parties receive each message block with the same value, we use a universal hash function similar to the protocol in [7]. Once a party receives a block from the designated party, it will randomly generate and broadcast a universal hash function key. The original sender P_s will respond by broadcasting the hash value of the block. All parties holding a block will also compute the hash values of their own blocks and compare to the value broadcast by P_s. They then broadcast whether or not the values are the same. Unlike in [7], multiple sessions of this correspondence can happen in parallel—one for each pair of parties transmitting a block. In order to guarantee that blocks received by multiple honest parties in the same round have the same value, we also require parties that just receive blocks to broadcast their hash checking result as well.

Trust Graph. We combine and expand the techniques for keeping track of a party's interactions in [7] and [6]. As in [7], each party collectively keeps track of conflict between each pair of parties. A conflict occurs between two parties P_a and P_b—both holding a block with P_b receiving a block from P_a earlier in the protocol—if one approves a hash value from P_s while another rejects it. In [6], each party instead keeps track of a set of corrupted parties from their own perspective. In both constructions, a party only tries to obtain a block from another party if it is not in conflict with that party or the party is not corrupted. We expand this idea to the concept of the public trust graph. A trust graph starts as a complete graph where vertices are all parties. When a pair of parties are in conflict in the same sense as in [7], an edge between them is removed. If a party publicly does not follow the protocol, it will be isolated in the trust graph. While the conflict set in [7] can be directly translated to our trust graph, we make additional use of the graph property to strengthen our protocol.

Condition to Forfeit a Block. Unlike the collision-resistant hash function used in [6], a universal hash function cannot be computed once and for all. If an adversary knows a hash key before it chooses whether to send a block, it can find a different block that hashes to the same value. In order to get around this limitation, the protocol in [7] lets the receiver choose a new hash key after it receives a block via point-to-point channel.

However, the verification in [7] is done separately for each receiver. In the situation where the sender P_s and block holders P_a and P_b collude, they can approve two different block values for honest P_i and P_j, who receive blocks from P_a and P_b, respectively. When P_j learns of the conflict between P_a and P_i, it cannot tell which of P_a and P_i is corrupted. In this case, P_j removes an edge $\{P_a, P_i\}$ from its trust graph. Since the conflict is known to every honest party via broadcast, the honest parties can maintain a consistent trust graph locally. In [7], whenever such conflict occurs, P_j must forfeit the block it has. Any pairs of parties in conflict do not send or receive a block from one another ever again across all message blocks. This guarantees that any two honest parties hold message blocks with the same value. As in [7], this means that each honest party may need to receive a block more than once. Since the trust graph has

$\mathcal{O}(n^2)$ edges, such a conflict can occur at most $\mathcal{O}(n^2)$ times. By dividing the message into blocks appropriately, [7] can keep the communication complexity to the optimal $\mathcal{O}(\ell n + poly(n, \lambda))$. However, our parallel block sending further increases the number of such forfeits as more than one party may try to get a block and fail at the same time. We solve this problem by implementing a stronger condition for a party to forfeit a block. Namely, P_j only forfeits a block when there is no trust path of block holders from P_j to P_s. Together with the next technique to increase the number of such paths, we can also keep the communication complexity the same as in [7].

Condition to Receive a Block. In order to reduce the number of forfeits which leads to an increase in communication complexity, we add additional conditions for when a party is to be sent a block. The idea is to make it harder for an adversary to force a party, who has already received a block, to forfeit it in a later round. The protocol in [7] uses a tree with P_s as a root to represent how a block is sent between parties. However, their protocol entirely resets this tree whenever a conflict occurs. Doing so, along with the parallel block sending technique, leads to an increase in both round complexity and communication complexity by a factor of n. Our first solution is, instead of resetting the tree, to disconnect the pair in conflict and remove those no longer connected to P_s. Unfortunately, this does not solve the problem. An adversary can still force a long path between P_s and honest parties, and repeatedly disconnect them from P_s. Instead, our protocol uses a graph H^j to represent the connection for jth block. H^j is an induced subgraph of the trust graph G on a subset of parties that have received a block. Due to the verification via universal hash function, all honest parties in H^j hold a block with the same value. When a party P_i is added to H^j, we add all edges between P_i and all parties in H^j that connect to P_i in G as well. Thus, in order for a party to be removed from H^j—which is equivalent to forfeiting a block—all of its neighbors in H^j need to be removed as well.

Varying Block Size. Our protocol takes $\mathcal{O}(d_j + \Delta_j)$ rounds to broadcast the jth block, where d_j is the maximum distance between the sender and receiving parties in the trusted graph and Δ_j is the number of edges removed from the graph while broadcasting the block. If the blocks are of the same size either ℓ/n^2, as in [7], or ℓ/n, as in [6], the resulting protocol will provide no improvement in round complexity. We solve this problem by using a non-constant block size of $\ell d_{j-1}/n^2$. Since $1 \leq d_{j-1} \leq n$, our block size is between that of [7] and [6]. In the case of an honest sender, $d_j = 1$ for all j, we get the same block size as in [7]. Intuitively, as the corrupted parties are known and the distance from receiving parties in G grows, we want to send a larger block because the number of edges that can be disconnected is smaller. It is more difficult for the corrupted parties to make the honest parties resend a block.

2 Definitions

Let λ denote the security parameter. A *negligible* function $\nu(\lambda)$ is a non-negative function such that for any constant $c < 0$ and for all sufficiently large λ, $\nu(\lambda) < \lambda^c$. We will denote by $\Pr_r[X]$ the probability of an event X over coins r, and $\Pr[X]$ when r is not specified. For a randomized algorithm A, let $A(x; r)$ denote running A on an input x with random coins r. If r is chosen uniformly at random with an output y, we denote $y \leftarrow A(x)$. Let $\mathcal{P}$ be a set of n parties $\{P_1, \ldots, P_n\}$. For a finite subset $A \subset U$, let $\overline{A}$ denote $U \setminus A$ when U is clear from context. For a vertex v of a graph G, we may use $v \in G$ to denote $v \in V(G)$.

Definition 1 (Byzantine Broadcast). *A protocol Π for a set of n parties $\mathcal{P}$, with secure private channel between every pair of parties, and a distinguished party P_s for some $s \in [n]$, called a sender, who holds an input $m \in \mathcal{M}$, is a secure* (Byzantine) broadcast *protocol if, at the end of the protocol, the following holds except with negligible probability:*

- *All honest parties output the same value $m' \in \mathcal{M} \cup \{\perp\}$; and*
- *If the sender P_s is honest, $m' = m$.*

Definition 2 (Universal Hash Function). *A family of functions $\{H_k\}_{k \in S_H}$ where $H_k : \mathcal{M} \to Y$ is ϵ-*universal *if for any two distinct $m, m' \in \mathcal{M}$,*

$$\Pr[k \leftarrow S_H : H_k(m) = H_k(m')] \leq \epsilon.$$

A universal hash function can be constructed as follows. Let $S_H = Y = \mathbb{F} = \mathbb{F}_{2^\lambda}$. Let $m \in \mathcal{M} = \{0,1\}^\ell$ be represented by a polynomial $m(x)$ over $\mathbb{F}$ by cutting m in blocks of size λ. We compute $H_k(m) = m(k) \in Y$.

3 Broadcast Extension

In this section we give an overview of the broadcast constructions of [7] and [6].

3.1 Information-Theoretic Secure Broadcast in $\mathcal{O}(n^4)$ Rounds

We first describe the broadcast extension protocol of [7]. Informally, the sender P_s cuts a long message into blocks. The protocol broadcasts each block sequentially using ITBlockBC. The subprotocol ITBlockBC works as follows. In each loop, a party P_a who has the block sends it to another party P_b that has not received it. P_b then generates and broadcasts a key k for information-theoretically secure universal hash function. Next, P_s computes and broadcasts the hash value of the block using the received key. Every party that has the block responds as to whether the block they have gives the same hash value. If there is a pair of parties P_c and P_d where P_d has received a block from P_c and the two disagree on the hash value, the subprotocol is restarted and $\{P_c, P_d\}$ is added to a “dispute set” where they will not interact again. This set is kept across multiple executions of ITBlockBC—one for each block. Thus, the conflict can only occur at most $\mathcal{O}(n^2)$ times across the executions. When no such conflict occurs, each execution of ITBlockBC takes $\mathcal{O}(n)$ rounds with oracle access to (short) broadcast. By cutting the message into n^2 blocks, the protocol gives $\mathcal{O}(n^3)$ rounds with the oracle access, and $\mathcal{O}(n^4)$ rounds when the oracle is substituted by an $\mathcal{O}(n)$-round broadcast protocol of [15].

Let $\{H_k\}_{k \in S_H}$ be a family of universal hash functions with seeds in S_H. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of all parties. We describe the protocol ITBlockBC in Fig. 1.

```
                         $\mathsf{ITBlockBC}(P_s, m)$
For each party $P_i$ on input dispute set $\Delta$.

 1. Initialize a set $H = \{P_s\}$ and $T = \emptyset$.
 2. While $\exists P_x, P_y \in \mathcal{P}$ such that $P_x \in H$, $P_y \in \overline{H}$ and $\{P_x, P_y\} \notin \Delta$ do the
    following:
 Round 1: $P_x$ sends $m_x$ to $P_y$ via point-to-point channel. $P_y$ sets $m_y := m_x$. Add
          $(P_x, P_y)$ to $T$.
 Round 2: $P_y$ generates and broadcasts $k \leftarrow S_H$.
 Round 3: $P_s$ broadcasts $h := H_k(m)$.
 Round 4: $\forall P_i \in H \cup \{P_y\} \setminus \{P_s\}$ if $h = H_k(m_i)$ broadcasts 1; otherwise, 0.
 Round 5: If all parties broadcast 1, add $P_y$ to $H$. Else,
            • for all $(P_i, P_j) \in T$ such that $P_i$ broadcast 1 or $P_i = P_s$ and $P_j$
              broadcast 0, add $\{P_i, P_j\}$ to $\Delta$; and
            • set $H = \{P_s\}$ and $T = \emptyset$.
 3. $\forall P_i \in \mathcal{P}$, if $P_i \in H$, output $m_i$; otherwise, output $\perp$.
```

Fig. 1. Information-theoretic block broadcast of [7]

The broadcast protocol can be obtained by running ITBlockBC n^2 times as shown in Fig. 2.

```
                         $\mathsf{LongBC}(P_s, m)$
 1. Parties initialize dispute set $\Delta = \emptyset$.
 2. Sender $P_s$ cuts $m$ into $n^2$ equal pieces $m^1, \ldots, m^{n^2}$ (padding if required).
 3. For $c = 1, \ldots, n^2$, invoke $\mathsf{ITBlockBC}(P_s, m^c)$ and let $m_i^c$ be the output of $P_i$.
 4. For each $P_i \in \mathcal{P}$, if $m_i^j = \perp$ for some $j$, output $\perp$. Otherwise, output
    $m_i^1 || \ldots || m_i^{n^2}$.
```

Fig. 2. Broadcast extension using ITBlockBC

Theorem 2 ([7]). *Assuming an oracle for broadcasting short messages, there exists a broadcast protocol* LongBC *achieving information-theoretic security in* $t < n$ *setting for an* ℓ*-bit message in* $\mathcal{O}(n^3)$ *rounds by communicating* $\mathcal{O}(\ell n + n^3(B(\lambda) + nB(1)))$ *bits.*

Corollary 2 ([7]). *There exists a broadcast protocol achieving information-theoretic security in* $t < n$ *setting for an* ℓ*-bit message in* $\mathcal{O}(n^4)$ *rounds by communicating* $\mathcal{O}(\ell n + n^{10}\lambda)$ *bits.*

3.2 Computationally Secure Broadcast in $\mathcal{O}(n^2)$ Rounds

The construction of [6] improves on the computational case of [7]. In [7] a long message is broadcast in blocks similar to the information-theoretic case above. Instead of generating a new key for universal hash function every time a party receives a block, the sender P_s broadcasts a hash value of the block using collision-resistant hash function (CRHF) at the beginning of the subprotocol. When a party P_b receives a block from P_a, he can verify it locally with no additional interaction. If the verification fails, P_b knows that P_a is corrupted. Thus, the failure can occur at most $\mathcal{O}(n)$ times. By cutting the message into n blocks, the protocol gives $\mathcal{O}(n^2)$ rounds with the oracle access, and $\mathcal{O}(n^3)$ rounds when the oracle is substituted by $\mathcal{O}(n)$-round broadcast protocol of [15].

In [6] this protocol is improved by allowing multiple parties to send and receive a block in the same round. Several checks are added to ensure that this parallel process does not break the correctness and security. This technique speeds up the protocol by a factor of n.

Let Hash be a collision-resistant hash function. We describe the protocol CryptoBC in Fig. 3.

Theorem 3 ([6]). *Assuming an oracle for broadcasting short messages and CRHFs, there exists a broadcast protocol* CryptoBC *against a PPT adversary corrupting* $t < n$ *parties for an* ℓ*-bit message in* $\mathcal{O}(n)$ *rounds by communicating* $\mathcal{O}(\ell n + (n\lambda + n^3 \log n)B(1))$ *bits.*

Corollary 3 ([6]). *Assuming CRHFs, there exists a broadcast protocol against a PPT adversary corrupting* $t < n$ *parties for an* ℓ*-bit message in* $\mathcal{O}(n^2)$ *rounds by communicating* $\mathcal{O}(\ell n + n^6 \lambda \log n)$ *bits.*

4 Our Construction

In this section we show how to improve information-theoretic secure broadcast for long messages in [7]. In [6], Ganesh et al. show that it is possible to broadcast a message of arbitrary length ℓ using $\mathcal{O}(n)$ rounds having $\mathcal{O}(n)$ black-box access to a broadcast protocol for single bit, assuming CRHF. Thus, combining the result with [4] gives a broadcast protocol for a message of arbitrary length in $\mathcal{O}(n^2)$ rounds under the same assumption. On the other hand, the best result for information-theoretic broadcast for arbitrary long messages by [7] uses $\mathcal{O}(n^3)$

```
$\mathsf{CryptoBC}(P_s, m)$
Hash Agreement phase:

1. $P_s$ cuts $m$ into $n$ equal pieces $m^1, \ldots, m^n$ (padding if required).
2. For $c = 1, \ldots, n$, $P_s$ computes and broadcasts $h^c = \mathsf{Hash}(m^c)$ to all parties.

Block Agreement phase: For each party $P_i$

1. Initialize
   – $C_i = \emptyset$, $c_i = 1$, $r = 1$;
   – $T_i^k[j, l] = 1$ for $j, l, k \in [n]$;
   – $H_i^k = \{P_s\}$ for $k \in [n]$;
2. While $r \leq n + t$ do
   (a) If $P_i \in \overline{H}_i^{c_i}$, $\exists P_j \in H_i^{c_i} \setminus C_i$ and $|H_i^{c_i} \cup C_i| \geq r - c_i + 1$, broadcast
       $(\mathsf{send}, j, c_i)$.
   (b) Let $(\mathsf{send}, x, y)$ be the output of the broadcast from $P_j \notin C_i$.
       i. if $T_i^y[x, j] = 1$ and there is only one broadcast from $P_j$, then set
          $T_i^y[x, j] = 0$, and if $x = i$ and $P_i \in H_i^y$, send $m_i^y$ to $P_j$ via point-to-
          point channel;
       ii. else, add $P_j$ to $C_i$.
   (c) If $P_i$ broadcast $(\mathsf{send}, j, c_i)$ in Step 2(a), let $\overline{m}_j^{c_i}$ be the message block
       received from $P_j$
       i. if $h^{c_i} = \mathsf{Hash}(\overline{m}_j^{c_i})$, then increment $c_i$ by 1, set $m_i^{c_i} = \overline{m}_j^{c_i}$ and broad-
          cast $(\mathsf{happy}, H_i^{c_i}, C_i, c_i)$;
       ii. else, broadcast $(\mathsf{unhappy}, c_i)$ and add $P_j$ to $C_i$.
   (d) Let $v$ be the output of the broadcast from $P_j \notin C_i$ in Step 2(c) who
       broadcast $(\mathsf{send}, \star, \star)$ in Step 2(a) this round
       i. if $v = (\mathsf{happy}, H_j^x, C_j, x)$, $H_j^x \cup C_j \subseteq H_i^x \cup C_i$ and $|H_j^x \cup C_j| \geq r - x + 1$,
          then add $H_j^x \cup \{P_j\}$ to $H_i^x$;
       ii. if $v = (\mathsf{unhappy}, x)$ do nothing;
       iii. else, add $P_j$ to $C_i$.
   (e) If $r = c_i + t$ and $P_i \in \overline{H}_i^{c_i}$, then exit while loop.
3. If $m_i^k = \perp$ for some $k \in [n]$, output $\perp$. Otherwise, output $m_i^1 || \ldots || m_i^n$.
```

Fig. 3. Computationally secure broadcast of [6] against $t < n$ corruption

rounds having $\mathcal{O}(n^3)$ black-box access to a broadcast protocol for single bit. Thus, combining the result with [14,15] gives a broadcast protocol in $\mathcal{O}(n^4)$ rounds. We show that several techniques, including parallel block broadcast in [6], can be used to improve this result to $\mathcal{O}(n^3)$ rounds.

We first describe a protocol ImprovedBlockBC that broadcasts a block of a long message using an oracle broadcasting short messages. Besides the message block as an input of the sender, each party P_i maintains a trust graph G_i across executions of ImprovedBlockBC for all message blocks. While our trust graph and the dispute set in [7] provide similar information, our protocol takes into account some properties of graph such as the length of a shortest path between a pair of nodes. Finally, we describe our broadcast protocol ImprovedLongBC running ImprovedBlockBC as a subprotocol. This protocol is similar to LongBC (in Sect. 3) but with a varying number of blocks depending on the state of the trust graph at the end of each execution of ImprovedBlockBC.

4.1 Improved Block Broadcast

The protocol ImprovedBlockBC modifies ITBlockBC (in Sect. 3) using several techniques. Similar to ITBlockBC, each party uses a universal hash function to verify whether a block it receives is "correct"—meaning that all honest parties agree on the value of the message block. To speed up the protocol, it also employs some of the parallel processing technique in [6]. Similar to CryptoBC of [6], ImprovedBlockBC allows multiple pair of parties to send and receive blocks at the same time. Additional conditions are checked to ensure that all honest parties agree on which parties sending and receiving blocks at all time. When all parties follow the protocol honestly, every party receives the block concurrently and ImprovedBlockBC terminates in $\mathcal{O}(1)$ round (with oracle access to short broadcast). On the other hand, ImprovedBlockBC operates on one block at time, unlike CryptoBC where different pairs of parties may send and receive different blocks at the same time. This is unavoidable due to the weaker guarantee of universal hash functions compared to that of collision-resistant hash functions.

We replace the dispute set Δ, the set H of parties that have already received a block, and the history set T with a trust graph G_i and a graph H_i. While they contain the same information, we utilize the graph properties including connectivity and path length in our protocol. Similar to ITBlockBC, a party may forfeit a block due to conflict in universal hash value. Instead of resetting the block broadcast entirely as in ITBlockBC—which can lead to larger round complexity—we minimize the number of such forfeits using two techniques. First, a party P_j is only sent a block when all of its neighbors in the trust graph that are closer to the sender already have the block. (In that case, we say P_j is "ready to receive a block.") Second, P_j only forfeits a block if it is disconnected to P_s in H_i, which only occurs when all of the neighbors above are also disconnected.

Let $\{H_k\}_{k \in S_H}$ be a family of universal hash functions with seeds in S_H. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of all parties with fixed ordering, e.g., $P_1 > P_2 > \ldots > P_n$. Each party P_i keeps its trusted graph G_i, where each node represents a party in $\mathcal{P}$, throughout ImprovedBlockBC for all message blocks. In the beginning of the first block, G_i is initialized to a complete graph C_n. If a broadcast protocol from P_a fails, P_i isolates P_a in G_i by removing all edges connecting to P_a. Let $G(P_s)$ denote the connected component of G containing P_s. We describe the protocol ImprovedBlockBC in Fig. 4. Note that all broadcasts in the same step can be done in parallel. P_i ignores all messages it does not expect as specified by the protocol.

Definition 3. *Let G be a graph on $\mathcal{P}$ and $H \subseteq G(P_s)$. We say P_j is* ready to receive a block from P_i with respect to (H, G, P_s) *if all of the following holds:*

- *P_j is a neighbor of P_i in G_i;*

- *$P_j \notin H$;*
- *For every shortest path from P_j to P_s, $(P_j, P_{j_k}, \ldots, P_s)$, $P_{j_k} \in H$;*
- *P_i is the maximal such P_{j_k} (with respect to the ordering given above).*

Now we prove the following properties of ImprovedBlockBC. The following lemma shows that G_i and H_i of honest parties are the same as they are only updated using information that is broadcast.

Lemma 1. *Suppose all honest parties hold the same G_i at the beginning of* ImprovedBlockBC*. Then, at the end of each while loop, all honest parties hold the same G_i and H_i.*

Proof. Assuming all honest parties hold the same G_i and H_i at the beginning of a while loop. Then in Round 1, 2 and 3, all honest parties agree whether P_y is ready to receive a block from P_x. Then, by the agreement property of broadcast, all honest parties agree on edge removal of G_i in Round 3 and hold the same recording (k_y, P_x, P_y)'s. Also by the agreement property, all honest parties agree on edge removal of G_i in Round 4 and 5. Finally, by the agreement property and the consistency of G_I, they also agree on modification of H_i in Round 5. Since the honest parties hold the same G_i and initialize the same H_i at the beginning of the protocol, the consistency of G_i and H_i holds at the end of each while loop. □

From this point onward, we assume all honest parties hold the same G_i at the beginning of ImprovedBlockBC, and denote the same G_i and H_i for all honest P_i by G and H, respectively. The following lemma shows the consistency of the values hold by honest parties. We use the property of universal hash functions when the keys are chosen uniformly at random by honest parties.

Lemma 2. *Except with negligible probability, at the end of each while loop, all honest parties in H hold the same value m.*

Proof. Assume that at the beginning of a loop, all honest parties in H hold the same value m. Suppose P_i is an honest party added to H in this loop. The statement holds trivially if there is no other honest party in H at the end of the loop. Suppose there is another honest party P_j in H at the end of the loop. Then P_i broadcasts (k_j, P_a) for some $P_a \in H$ in Round 2 and P_s broadcasts (h_i, P_i) in Round 3. Also, P_j broadcasts (true, P_i) in Round 4; otherwise, P_j would be removed from or not added to H_i. Since P_i and P_j are honest $H_{k_i}(m_a) = H_{k_i}(m_j) = h_i$. By the property of universal hash function, since k_i is chosen honestly independent of the messages, except with negligible probability, $m_a = m_j = m$. The result follows as the first loop has $V(H) = \{P_s\}$. □

Let Good be the event that, at the end of each while loop, all honest parties in H hold the same value m.

```
ImprovedBlockBC($P_s, m$)
For each party $P_i$ on input a trust graph $G_i$.

1. Initialize a graph $H_i \subseteq G_i(P_s)$ with only one vertex $P_s$ and no edge.
2. While $P_i \in G_i(P_s)$ and $|V(H_i)| < |V(G_i(P_s))|$, clear all records and do
   Round 1: If $P_i \in H_i$, for each $P_j$ ready to receive a block from $P_i$ with respect
            to $(H_i, G_i, P_s)$ $P_i$ sends $m_i$ to $P_j$ via point-to-point channel.
   Round 2: If $P_i \notin H_i$ and is ready to receive a block from $P_j$ with respect to
            $(H_i, G_i, P_s)$,
            (a) if $P_i$ does not receive $m_j$ or receive more than one block from $P_j$
                in Round 1, broadcast $(\mathsf{fail}, P_j)$ and remove $\{P_i, P_j\}$ from $E(G_i)$;
            (b) else, sample $k \leftarrow S_H$ and broadcast $(k, P_j)$ and record $(k, P_j, P_i)$.
   Round 3: When $P_i$ outputs $(A_y, P_x)$ broadcast by $P_y$, if $\{P_x, P_y\} \notin E(G_i)$ or $P_y$
            is not ready to receive a block from $P_x$ with respect to $(H_i, G_i, P_s)$,
            isolate $P_y$ in $G_i$. Else
            (a) if $A_y = \mathsf{fail}$, remove $\{P_x, P_y\}$ from $E(G_i)$;
            (b) if $A_y = k_y$, record $(k_y, P_x, P_y)$;
            (c) if $P_i = P_s$, broadcast $(H_{k_y}(m), P_y)$;
            (d) if $P_i$ receives multiple broadcast messages from $P_y$ this round or
                $A_y$ is not one of the above, isolate $P_y$ in $G_i$.
   Round 4: When $P_i$ outputs $(h_y, P_y)$ broadcast by $P_s$, if $(k_y, P_x, P_y)$ is not
            recorded, output $\perp$ and abort; else if $P_i \in H_i$ or received $m_j$ in Round
            1, check if $H_{k_y}(m_i) = h_y$ or $H_{k_y}(m_j) = h_y$, respectively. Broadcast
            $(\mathsf{true}, P_y)$ or $(\mathsf{false}, P_y)$ accordingly. If there exists a record $(k_y, P_x, P_y)$
            without $(h_y, P_y)$ broadcast, output $\perp$ and abort.
   Round 5: When $P_i$ outputs $(\mathsf{true}, P_y)$ or $(\mathsf{false}, P_y)$ with $(k_y, P_x, P_y)$ recorded
            broadcast by $P_b$ either in $H_i$ or with $(k_b, P_a, P_b)$ recorded, $P_i$ appends
            $(P_b, \mathsf{true}/\mathsf{false})$ to the recording $(k_y, P_x, P_y)$. Isolate any $P_b$ broadcasting
            both $(\mathsf{true}, P_y)$ and $(\mathsf{false}, P_y)$, or $P_b$ either in $H_i$ or with $(k_b, P_a, P_b)$
            recorded broadcasting neither. At the end of this round, $P_i$ processes
            each recorded $(k_y, P_x, P_y, \ldots)$ one by one in the order of $P_y$ as follows.
            (a) For each $P_b \in H_i$ whose $(P_b, \mathsf{false})$ is appended,
                i. for each $P_a$, $P_b$'s neighbor in $G_i$, if $(P_a, \mathsf{true})$ is appended (or
                   $P_a = P_s$), remove $\{P_a, P_b\}$ from $E(G_i)$ and $E(H_i)$;
                ii. remove $P_b$ from $H_i$.
            (b) For each $P_b$ with $(k_b, P_a, P_b, \ldots)$ recorded, if $(P_a, \mathsf{true})$ is appended
                (or $P_a = P_s$), remove $\{P_a, P_b\}$ from $E(G_i)$ and append $\mathsf{fail}$ to
                $(k_b, P_a, P_b, \ldots)$.
            (c) Ignore $P_b$ that is removed from $H_i$ earlier this round.
            After processing all records, remove any $P_a$ no longer connected to $P_s$
            in $H_i$ from $H_i$. For each recorded $(k_y, P_x, P_y, \ldots)$, if $P_x$ is still in $H_i$,
            $\{P_x, P_y\}$ is still in $E(G_i)$ and no $\mathsf{fail}$ appended, add $P_y$ to $V(H_i)$ and
            $\{P_x, P_y\}$ to $E(H_i)$ and if $P_i = P_y$, set $m_i = m_y$.
3. If $P_i \in H_i$, output $m_i$. Otherwise, output $\perp$.
```

Fig. 4. Improved block broadcast

Lemma 3. *Assuming the event* Good *occurs, for any two different honest parties P_i and P_j, $\{P_i, P_j\} \in E(G)$ at any point in the protocol. Furthermore, at the end of the protocol, either all honest parties are in H and output the same m, or output $\perp$.*

Proof. An honest P_a removes $\{P_i, P_j\}$ from $E(G)$ when one of the following holds:

1. P_j is ready to receive a block from P_i but does not get one or get more than one in Round 1 and broadcasts (fail, P_i) in Round 2;
2. P_j broadcasts malformed or multiple messages in Round 2;
3. P_i and P_j broadcast different $(\mathsf{true}/\mathsf{false}, P_y)$ with (k_y, P_x, P_y) recorded in Round 4.

By Lemma 1, the first two conditions do not occur for honest P_i and P_j. By Lemma 2, the last condition does not occur for honest P_i and P_j. Thus, $\{P_i, P_j\}$ is never removed from $E(G)$.

By the agreement property of broadcast, honest parties agree on the abort condition in Round 4. If the abort condition does not occur, the protocol ends when $P_i \notin G(P_s)$ or $|V(H)| = |V(G(P_s))|$. Since honest parties are connected in G, they agree on the first condition. The honest parties also agree on the second condition by Lemma 1, and if the first condition does not hold, it implies $P_i \in H = G(P_s)$ for all honest P_i. By Lemma 2, they all output m. □

Let G^* be the trust graph G at the end of the protocol. Let $H^* = G^*(P_s)$. We let $d(P_i)$ denote the length of the shortest path from P_i to P_s in H^* and $d = \max_i d(P_i)$. For $j = 1, \ldots, d$, let Δ_j be the number of edges removed from G when all parties P_i with $d(P_i) \leq j$ are last added to H (i.e., not removed later in the protocol). We have $0 \leq \Delta_1 \leq \Delta_2 \leq \ldots \leq \Delta_d \leq \Delta$.

Lemma 4. *Suppose a party P_i is in H at the end of the protocol, then P_i is last added to H in $t_i = t(P_i) \leq d(P_i) + \Delta_{d(P_i)}$ loops. In particular, assuming the event* Good *occurs, the protocol ends in $5(d + \Delta)$ rounds.*

Proof. We prove the statement by induction on $d(P_i)$. Clearly, when $d(P_i) = 1$, $(P_i, P_s) \in E(G^*)$ and P_s sends a block to P_i every loop until P_i is added to H. If P_i fails to be added, a neighbor of P_i broadcasts (false, P_i), and thus there must be an edge (P_a, P_b) that gets removed from $E(G)$. Thus, $t_i = t(P_i) \leq d(P_i) + \Delta_1$. Suppose any P_j with $d(P_j) = d(P_i) - 1$ is last added to H in $t_j \leq d(P_j) + \Delta_{d(P_j)} = d(P_i) + \Delta_{d(P_j)} - 1$ loops. In the $(t_j + 1)$th loop, either $P_i \in H$ or $P_i \notin H$. Suppose $P_i \in H$. Then P_i is not removed in or after this loop as P_j is not. Otherwise, a neighbor P_j of P_i on the shortest path will have to be added after t_jth loop, which is a contradiction. Thus, $t_i \leq t_j \leq d(P_i) + \Delta_{d(P_i)}$. Now suppose $P_i \notin H$. Then P_i is ready to receive a block from one of its neighbors every loop after t_j as all of its neighbors on the shortest path are in H and have never been removed. Thus, in every loop after t_j, either P_i gets a block or an edge gets removed from $E(G)$. Therefore, $t_i = t_j + 1 + (\Delta_{d(P_i)} - \Delta_{d(P_i)-1}) \leq d(P_i) + \Delta_{d(P_i)}$.

Now assume that the event Good occurs. If an honest party is in H at the end of the protocol, then by Lemma 3, all honest parties are in H at the end of the protocol. The last honest party is last added to H in $d + \Delta$ loops, i.e., $5(d + \Delta)$ rounds. Otherwise, suppose all honest parties are not in H at the end of the protocol. By Lemma 3 and the agreement property of broadcast, honest parties terminate at the same time. Let t^* be the last loop before the termination. Suppose that every party follows the protocol correctly from the next loop onward. Then an honest party P_i will stay in $G_i(P_s)$ and be added to H within $t' \leq d + \Delta$ loops. We have $t^* \leq t' \leq d + \Delta$ as well. □

Lemma 5. *Let d_0 be the maximum length of the shortest path from any honest party to P_s at the beginning of the protocol. Let d_1 be the maximum length of the shortest path from any honest party to P_s at the end of the protocol. The number of times a block is sent to and from honest parties is at most $\mathcal{O}(n+\Delta+n(d_1-d_0))$.*

Proof. Every party in $G^*(P_s)$ must receive a block at least once. Thus, we need n times. A party receives a block more than once under two conditions:

1. $P_j \notin H$ is ready to receive a block from P_i but fails due to
   (a) P_i does not send a block; or
   (b) (P_i, P_j) is removed from $E(G)$; or
   (c) P_i is removed from H.
2. $P_j \in H$ is removed from H.

For 1(a) and 1(b), $|E(G)|$ decreases by 1. For 1(c) and 2, the shortest path of some party increases by at least 1. Thus, the number of additional times a party needs to get a block is bounded by $\Delta + n(d_1 - d_0)$. □

4.2 Improved Broadcast Extension

Now we are ready to describe our main construction of broadcast extension using block broadcast ImprovedBlockBC as a subprotocol. As in [7], in order to broadcast a message m of arbitrary length ℓ, we cut m into q blocks. Unlike in [7], the block size will vary depending on the trust graph G at the end of the previous block. In particular, each block m_j has length $\ell_j = \ell d_{j-1}/n^2$ where d_j is the maximum length of the shortest path from P_s to any P_i connected to P_s at the end of jth execution of ImprovedBlockBC. We let $d_0 = 1$ and allow the last block to be shorter so that the length of all q blocks add up to ℓ. We then run ImprovedBlockBC in Fig. 4 q times sequentially as shown in Fig. 5.

Now we prove the round complexity and communication complexity of ImprovedLongBC. Let d_j be the maximum length of the shortest path from P_s to any P_i connected to P_s at the end of jth execution of ImprovedBlockBC, and Δ_i be the decrease in number of edges of G.

Lemma 6. *Assuming an oracle for broadcasting short messages,* ImprovedLongBC *takes at most $\mathcal{O}(n^2)$ rounds.*

```
ImprovedLongBC($P_s, m$)
1. Each party $P_i$ initializes a trust graph $G_i = C_n$, a complete graph on $V = \mathcal{P}$.
2. Sender $P_s$ initializes $m_1$, the first $\ell_1$ bits of $m$ where $\ell_1 = \ell/n^2$ (padding if $\ell$
   is not divisible by $n^2$), and sets $c = 1$.
3. While $\sum_{j=1}^{c} \ell_j < \ell$, do the following:
   (a) Invoke ImprovedBlockBC($P_s, m_c$) and let $m_c^i$ be the output of $P_i$.
   (b) If $|m_c^i| \neq \ell_c$, $P_i$ aborts.
   (c) Compute $d_c$ the maximum length of the shortest path from $P_s$ to any $P_i$
       connected to $P_s$.
   (d) Let $\ell_{c+1} = \ell d_c/n^2$ and $m_{c+1}$ be the next $\ell_{c+1}$ bits of $m$.
   (e) increase $c$ by 1.
4. For each $P_i$, if $m_j^i = \perp$ for some $j$, output $\perp$. Otherwise, output $m_1^i || \ldots || m_q^i$
   where $q$ is the number of ImprovedBlockBC invoked.
```

Fig. 5. Broadcast extension using ImprovedBlockBC

Proof. The round complexity of LongBC is the sum of the round complexity of ImprovedBlockBC. By Lemma 4, the round complexity is

$$\sum_{j=1}^{q} 5(d_j + \Delta_j) = 5\left(\sum_{j=1}^{q} d_j + \sum_{j=1}^{q} \Delta_j\right)$$

Since $\ell = \sum_{j=1}^{q} \ell_j = \ell(1 + \sum_{j=1}^{q-1} d_j)/n^2$,

$$\sum_{j=1}^{q} d_j = n^2 - 1 + d_q \leq n^2 + n - 1$$

and $\sum_{j=1}^{q} \Delta_j \leq |E(C_n)| \leq n^2/2$. We have the round complexity $\mathcal{O}(n^2)$. □

Let d'_j be the maximum length of the shortest path from P_s to any *honest* P_i connected to P_s at the end of jth execution of ImprovedBlockBC. Let $B(l)$ be the communication complexity of broadcasting l bits.

Lemma 7. *The number of bits sent and received by honest parties in* ImprovedLongBC *is at most* $\mathcal{O}(\ell n + n^3(B(\lambda) + nB(\log n)))$.

Proof. The communication complexity of ImprovedLongBC is the sum of the communication complexity of ImprovedBlockBC.

By Lemma 5, the number of blocks communicated is $b_j \leq n + \Delta_j + n(d_j - d_{j-1})$ where each block incurs the communication of $\ell_j + B(|k| + \log n) + B(|h| + \log n) + nB(1 + \log n)$. Thus, the communication complexity is

$$\sum_{j=1}^{q} b_j \ell_j + \sum_{j=1}^{q} b_j \left(B(|k| + \log n) + B(|h| + \log n) + nB(1 + \log n)\right)$$

The first sum is

$$\begin{aligned}\sum_{j=1}^{q} d_j\ell_j &\leq \sum_{j=1}^{q} (n + \varDelta_j + n(d_j - d_{j-1}))\,\frac{\ell d_{j-1}}{n^2}\\ &= \ell\left(\frac{\sum_{j=1}^{q} d_{j-1}}{n} + \frac{\sum_{j=1}^{q} \varDelta_j d_{j-1}}{n^2} + \frac{\sum_{j=1}^{q} d_{j-1}(d'_j - d'_{j-1})}{n}\right)\\ &\leq \ell\left(\frac{n^2}{n} + \frac{n\sum_{j=1}^{q} \varDelta_j}{n^2} + \frac{\left(\sum_{j=1}^{q} d_{j-1}\right)\left(\sum_{j=1}^{q}(d'_j - d'_{j-1})\right)}{nq}\right)\\ &\leq \ell\left(n + n + \frac{n^3}{nq}\right) \leq 3\ell n.\end{aligned}$$

as $d'_j \leq d_j \leq n$ and $q \geq n$. Since $\sum_{j=1}^{q} b_j \leq nq + \sum_{j=1}^{q} \varDelta_j + nd'_q \leq n^3 + 2n^2$, the communication complexity is

$$\begin{aligned}&3\ell n + (n^3 + 2n^2)\,(B(|k| + \log n) + B(|h| + \log n) + nB(1 + \log n))\\ &\quad = \mathcal{O}(\ell n + n^3(B(\lambda) + nB(\log n))).\end{aligned}$$

□

Since the correctness of ImprovedLongBC follows directly from the correctness of ImprovedBlockBC from Lemma 2 and 3, we get the following theorem.

Theorem 4. *Assuming an oracle for broadcasting short messages, there exists a broadcast protocol achieving information-theoretic security in $t < n$ setting for an ℓ-bit message in $\mathcal{O}(n^2)$ rounds by communicating $\mathcal{O}(\ell n + n^3(B(\lambda) + nB(\log n)))$ bits.*

Combining the above result with the broadcast protocol of [15] gives the following corollary.

Corollary 4. *There exists a broadcast protocol achieving information-theoretic security in $t < n$ setting for an ℓ-bit message in $\mathcal{O}(n^3)$ rounds by communicating $\mathcal{O}(\ell n + n^{10}\lambda)$ bits.*

This result improves round complexity from instantiating the broadcast extension protocol in [7] with the broadcast protocol in [15] while maintaining the communication complexity.

5 Conclusion

We studied the broadcast protocols for long messages in the $t < n$ setting with the information-theoretic security. We modify and improve the broadcast extension protocol in [7], with the previously best-known round complexity of $\mathcal{O}(n^3)$ assuming an oracle for short messages. Our broadcast extension protocol has round complexity of $\mathcal{O}(n^2)$ while maintaining the same communication complexity. Combining our result with the broadcast protocol of [15] gives a broadcast extension protocol in the $t < n$ setting that achieves the communication complexity $\mathcal{O}(\ell n + n^{10}\lambda)$ and the round complexity of $\mathcal{O}(n^3)$. We leave an open question on how to further improve the round complexity to $\mathcal{O}(n^2)$ matching the computational case in [6] or to the optimal round complexity of $\mathcal{O}(n)$.

References

1. Beaver, D.: Correlated pseudorandomness and the complexity of private computations. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 479–488. ACM (1996)
2. Chaum, D., Roijakkers, S.: Unconditionally-secure digital signatures. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 206–214. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-38424-3_15
3. Dolev, D., Reischuk, R.: Bounds on information exchange for byzantine agreement. J. ACM (JACM) **32**(1), 191–204 (1985)
4. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
5. Fitzi, M., Hirt, M.: Optimally efficient multi-valued byzantine agreement. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, pp. 163–168. ACM (2006)
6. Ganesh, C., Patra, A.: Optimal extension protocols for byzantine broadcast and agreement. IACR Cryptol. ePrint Arch. **2017**, 63 (2017)
7. Hirt, M., Raykov, P.: Multi-valued byzantine broadcast: The $t<n$ case. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 448–465. Springer, Berlin (2014). https://doi.org/10.1007/978-3-662-45608-8_24
8. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_9
9. Liang, G., Vaidya, N.: Error-free multi-valued consensus with byzantine failures. In: Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pp. 11–20. ACM (2011)
10. Lindell, Y., Zarosim, H.: On the feasibility of extending oblivious transfer. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 519–538. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_29
11. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_40
12. Patra, A.: Error-free multi-valued broadcast and byzantine agreement with optimal communication complexity. In: Fernàndez Anta, A., Lipari, G., Roy, M. (eds.) OPODIS 2011. LNCS, vol. 7109, pp. 34–49. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25873-2_4
13. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM (JACM) **27**(2), 228–234 (1980)

14. Pfitzmann, B., Waidner, M.: Unconditional byzantine agreement for any number of faulty processors. In: Finkel, A., Jantzen, M. (eds.) STACS 1992. LNCS, vol. 577, pp. 337–350. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55210-3_195
15. Pfitzmann, B., Waidner, M.: Information-Theoretic Pseudosignatures and Byzantine Agreement for $t>n/3$. IBM, Armonk (1996)
16. Turpin, R., Coan, B.A.: Extending binary byzantine agreement to multivalued byzantine agreement. Inf. Process. Lett. **18**(2), 73–76 (1984)